DEVSUPERIOR

Formação Desenvolvedor Moderno
Módulo: Front end

Capítulo: Login e controle de acesso

https://devsuperior.com.br

Checklist: estruturar login e controle de acesso

1. Formulário e requisição de login (OAuth2)

2. Salvar token globalmente no localStorage *(para que possamos fazer requisições a recursos protegidos)*

3. Interceptors e redirecionamento fora do componente React *(para que possamos globalmente redirecionar quando o back end responder 401 ou 403)*

4. Fluxos de controle de acesso
    - 4.1. Controle de acesso em nível de rota
    - 4.2. Controle de acesso em nível de componente (restrição de conteúdo)

1. Formulário e requisição de login (OAuth2)

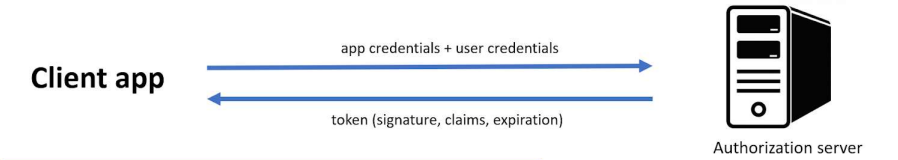
https://github.com/devsuperior/dscommerce-html-css

Biblioteca QS

```
yarn add qs@6.11.0 @types/qs@6.9.7
```

AuthService
+ loginRequest(loginData : CredentialsDTO) : Promise

Requisição de login (OAuth2)



App credentials
- **client_id**: myappname123
- **client_secret**: myappsecret123

Request Header:
Authorization: "Basic " + Base64.encode(client_id + ":" + client_secret)

User credentials
- **username**: nina@gmail.com
- **password**: 123456
- **grant_type**: password

Body: x-www-form-urlencoded

Token

JWT (RFC 7519 - jwt.io)

https://youtu.be/n1z9lx4ymPM

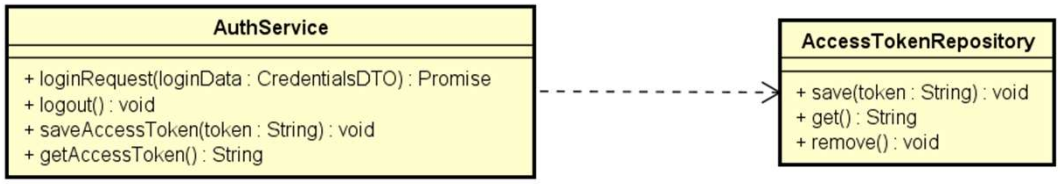
```
export function loginRequest(loginData: CredentialsDTO) {
  const headers = {
    "Content-Type": "application/x-www-form-urlencoded",
    Authorization: "Basic " + window.btoa(CLIENT_ID + ":" + CLIENT_SECRET),
  };

  const data = QueryString.stringify({
    ...loginData,
    grant_type: "password",
  });

  const config: AxiosRequestConfig = {
    method: "POST",
    url: "/oauth/token",
    data,
    headers,
  };

  return requestBackend(config);
}
```

5

5

2. Salvar token globalmente no localStorage

Objetivo: uma vez que o usuário realizou o login e obteve um token, precisamos armazenar este token localmente, para que possamos realizar requisições a recursos protegidos.



6

6

Requisições a recursos protegidos

Cabeçalho da requisição:

Authorization: Bearer <token>

```
export function requestBackend(config: AxiosRequestConfig) {
  const headers = config.withCredentials
    ? {
        ...config.headers,
        Authorization: "Bearer " + authService.getAccessToken()
      }
    : config.headers;

  return axios({ ...config, baseURL: BASE_URL, headers });
}
```


3. Interceptors e redirecionamento fora do componente React

Objetivo:

- Redirecionar para a tela de login caso algum componente receba resposta 401 do back end.
- Redirecionar para a tela de catálogo (ou outra) caso algum componente receba resposta 403 do back end.

Interceptors (Axios)

https://github.com/axios/axios#interceptors

```
// REQUEST INTERCEPTOR
axios.interceptors.request.use(
  function (config) {
    // DO SOMETHING BEFORE REQUEST IS SENT
    return config;
  },
  function (error) {
    // DO SOMETHING WITH REQUEST ERROR
    return Promise.reject(error);
  }
);
```

```
// RESPONSE INTERCEPTOR
axios.interceptors.response.use(
  function (response) {
    // DO SOMETHING WITH RESPONSE DATA IF STATUS IS 2xx
    return response;
  },
  function (error) {
    // DO SOMETHING WITH RESPONSE ERROR
    return Promise.reject(error);
  }
);
```


Redirecionamento fora do componente

```
yarn add history@5.3.0
```

```
import { createBrowserHistory } from 'history';

export const history = createBrowserHistory();
```

```
import { unstable_HistoryRouter as HistoryRouter } from 'react-router-dom';
import {history} from './utils/history';


<HistoryRouter history={history}>
   <Routes>
      ...
   </Routes>
</HistoryRouter>
```

4. Fluxos de controle de acesso

4.1. Controle de acesso em nível de rota

4.2. Controle de acesso em nível de componente (restrição de conteúdo)

4.1. Controle de acesso em nível de rota

Objetivo: permitir que o front end possa executar uma ação caso o usuário acesse uma rota que não tenha permissão, sem depender de um erro 401 ou 403 advindo do back end.

Faremos dois testes diferentes:

- Usuário autenticado (isAuthenticated)
- Usuário possui algum dos perfis informados (hasAnyRoles)

Teste: usuário autenticado

```
yarn add jwt-decode@3.1.2 @types/jwt-decode@3.1.0
```

AuthService:

```
export function getAccessTokenPayload(): AccessTokenPayloadDTO | undefined {
  try {
    const token = accessTokenRepository.get();
    return token == null ? undefined : (jwtDecode(token) as AccessTokenPayloadDTO);
  } catch (error) {
    return undefined;
  }
}

export function isAuthenticated(): boolean {
  let tokenPayload = getAccessTokenPayload();
  return tokenPayload && tokenPayload.exp * 1000 > Date.now() ? true : false;
}
```


Componente PrivateRoute

```
type Props = {
    children: JSX.Element;
}

export function PrivateRoute({ children }: Props) {
    if (!authService.isAuthenticated()) {
        return <Navigate to="/login" />;
    }
    return children;
}
```

```
<Route path="/admin/" element={<PrivateRoute><Admin /></PrivateRoute>} />
```

Teste: usuário possui algum dos perfis informados

AuthService:

```
export function hasAnyRoles(roles: RoleEnum[]): boolean {
  if (roles.length === 0) {
    return true;
  }

  const tokenPayload = getAccessTokenPayload();

  if (tokenPayload !== undefined) {
    for (var i = 0; i < roles.length; i++) {
      if (tokenPayload.authorities.includes(roles[i])) {
        return true;
      }
    }
    //return roles.some(role => tokenData.authorities.includes(role));
  }

  return false;
}
```


PrivateRoute atualizado

```
type Props = {
    children: JSX.Element;
    roles?: RoleEnum[];
}

export function PrivateRoute({ children, roles = [] }: Props) {
    if (!authService.isAuthenticated()) {
        return <Navigate to="/login" />;
    }
    if (!authService.hasAnyRoles(roles)) {
        return <Navigate to="/catalog" />;
    }
    return children;
}
```

```
<Route path="/admin/" element={<PrivateRoute roles={['ROLE_ADMIN']}><Admin /></PrivateRoute>}>
```

4.2. Controle de acesso em nível de componente

Objetivo: restringir conteúdo conforme o perfil do usuário

```
<div className="dsc-menu-items-container">
    {
        authService.hasAnyRoles(['ROLE_ADMIN']) &&
        <Link to="/admin">
            <div className="dsc-menu-item">
                <img src={iconAdmin} alt="Admin" />
            </div>
        </Link>
    }
    <Link to="/cart">
        <div className="dsc-menu-item">
            <CartIcon />
        </div>
    </Link>
</div>
```


Estado global do payload do token com Context API

```
export type ContextTokenType = {
    contextTokenPayload: AccessTokenPayloadDTO | undefined;
    setContextTokenPayload: (accessTokenPayload: AccessTokenPayloadDTO | undefined) => void;
}

export const ContextToken = createContext<ContextTokenType>({
    contextTokenPayload: undefined,
    setContextTokenPayload: () => {}
});
```

```
function App() {

  const [contextTokenPayload, setContextTokenPayload] = useState<AccessTokenPayloadDTO>();

  useEffect(() => {
    if (authService.isAuthenticated()) {
      const payload = authService.getAccessTokenPayload();
      setContextTokenPayload(payload);
    }
  }, []);

  return (
    <ContextToken.Provider value={{ contextTokenPayload, setContextTokenPayload }}>
```